МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторної роботи 1
за темою «Основні концепції програмування. Розробка та графічне подання алгоритмів»
з курсу «Алгоритмізація та програмування. Частина 1»

для студентів спеціальності
122 «Комп’ютерні науки»

Харків 2019

Методичні вказівки до виконання лабораторної роботи 1 за темою «Основні концепції програмування. Розробка та графічне подання алгоритмів» з курсу «Алгоритмізація та програмування. Частина 1» для студентів спеціальності 122 «Комп'ютерні науки» / уклад. Л. В. Іванов, М. О. Білова. – Харків : НТУ «ХПІ», 2019. – 29 с.

Укладачі: Л. В. Іванов
М. О. Білова

Рецензент Т. В. Козуля

Кафедра програмної інженерії та інформаційних технологій управління

ЗМІСТ

Вступ

Курс «Алгоритмізація та програмування» присвячений теоретичним та практичним аспектам розробки алгоритмів і програм мовою С++. Отримані в результаті вивчення даної дисципліни знання та навички можуть бути використані в усіх наступних курсах, під час виконання курсових та дипломних проектів, вивчення дисциплін, що пов’язані з обчислювальною технікою та програмуванням.

У процесі виконання лабораторної роботи за темою «Основні концепції програмування. Розробка та графічне подання алгоритмів» студенти мають закріпити теоретичний матеріал, отримати навички практичної роботи при реалізації циклічного алгоритму та алгоритму з розгалуженням на мові С++.

Перша частина присвячена теоретичним аспектам, пов’язаним з основними поняттями інформатики та програмування для здобуття студентами необхідного мінімуму компетентності у синтаксисі та семантиці мови програмування С++. Наведені конкретні приклади з побудови простих задач, задач з циклами.

Друга частина складається з завдань для лабораторної роботи, що використовуються як поточний контроль засвоєння матеріалу. Лабораторні заняття розраховані на роботу в комп’ютерному класі під безпосереднім керівництвом викладача та самостійну роботу студентів, яка передбачає написання тексту програми, закріплення практичних навичок роботи на комп’ютері, набутих при виконанні відповідної лабораторної роботи. Варіанти індивідуальних завдань видаються викладачем на занятті.

У методичних вказівках подано шістандцять індивідуальних варіантів завдань. У цілому методичні вказівки будуть корисні студентам різних спеціальностей і форм навчання, які вивчають мову С ++.

ТЕОРЕТИЧНА ЧАСТИНА

1. Основні поняття інформатики

Основні поняття інформатики пов’язані з системами числення, програмним забезпеченням, операційними та файловими системами, а також з особливостями використання текстових та бінарних файлів.

1.1. Системи числення

Система числення – це сукупність правил і знаків, за допомогою яких можна подати (закодувати) будь-яке число. Найбільш поширені **позиційні системи числення** – системи, в яких одна і та ж цифра у записі числа має різні значення залежно від позиції, в якій вона розташована.

У позиційній системі числення присутнє поняття «**основи системи числення»**. Будь-яке число може бути подано як сума ступенів основи, помножених на значення відповідних цифр. Значення основи дорівнює кількості цифр, що використовуються для запису числа. За основу системи числення можна прийняти будь-яке число, не менше ніж 2. Найменування системи числення відповідає її основі (десяткова, двійкова, вісімкова, шістнадцяткова тощо). У повсякденній практиці використовують **десяткову систему числення**.

В обчислювальній техніці традиційно прийнята **двійкова система числення**, заснована на поданні всієї інформації у вигляді логічних послідовностей нулів і одиниць. Ця система має дві основні переваги:

– для подання одиниці використовується електронний сигнал (рівень напруги), відмінний від нуля; оскільки амплітуда сигналу не має значення, це істотно підвищує надійність зберігання і передачі даних;

– правила виконання арифметичних операцій дуже прості й легко можуть бути реалізовані апаратно.

До недоліків двійкової системи можна віднести громіздкість запису чисел і недостатню наочність. У тих випадках, коли для роботи важливо саме двійкове подання числа, використовують системи числення з основою –

ступенем двох. Раніше була поширена вісімкова система, наразі найбільш часто використовується шістнадцяткова система числення. Для переведення із двійкової системи числення в шістнадцяткову кожні чотири двійкових розряди, починаючи справа, замінюються однією шістнадцятковою цифрою. В якості шістнадцяткових цифр прийнято використовувати десять десяткових цифр і букви A (10), B (11), C (12), D (13), E (14) і F (15). Зворотній переклад аналогічний – кожна шістнадцяткова цифра замінюється чотирма двійковими.

1.2. Програмне забезпечення

Термін «програма» використовують у двох значеннях:

– для опису **послідовності інструкцій**, які написані програмістом (сирцевого коду);

– для опису **одиниці програмного забезпечення**, яка може бути виконана на комп'ютері.

Програмне забезпечення (ПЗ, software) – це сукупність програм і програмних документів, необхідних для експлуатації цих програм. Загальноприйнятою є така класифікація програмного забезпечення:

1) системне ПЗ – програми, що забезпечують управління компонентами комп'ютерної системи; до системного ПЗ можна віднести:

– операційні системи (operating systems);

– системні утиліти (utilities);

– СУБД (системи управління базами даних, database management systems);

– драйвери пристроїв (device drivers);

2) прикладне ПЗ – програми і пакети програм, призначені для розв'язання задач користувачів у конкретних предметних галузях; до прикладного програмного забезпечення відносять текстові процесори, графічні редактори, поліграфічні системи, програми для виконання наукових і технічних розрахунків, ігри і т. д.;

3) інструментальні засоби – програмне забезпечення, призначене для створення іншого програмного забезпечення; до інструментальних засобів відносять компілятори та інші утиліти для збирання й налагодження, а також інтегровані середовища розробки (IDE, Integrated Development Environment) і CASE-системи (Computer-Aided Software Engineering).

Застосунок (додаток, застосування, application) – це синонім прикладної комп'ютерної програми. Однак часто під застосунком розуміють будь-яку програму, яка не є частиною операційної системи.

1.3. Операційні системи

Операційна система – це сукупність програмних засобів, що забезпечують управління всіма ресурсами і процесами обчислювальної системи. Під **ресурсами** розуміють процесор, пам'ять, дисплей, а також периферійні (зовнішні) пристрої. **Процеси** – це окремі програми, що отримали необхідні ресурси і запущені на виконання.

У складі операційної системи виділяють три групи компонентів:

– ядро (планувальник, драйвери пристроїв, підтримка мережі, файлова система);

– системні бібліотеки;

– набір утиліт.

Приклади операційних систем – MS-DOS, OS/2, Mac OS, MS Windows різних версій, UNIX, Linux, Solaris, Google Android і багато інших.

1.4. Файлова система

Файлова система (file system) – це спосіб і правила організації та зберігання каталогів і файлів на зовнішньому пристрої. Іноді під файловою системою також розуміють сукупність файлів і каталогів на конкретному пристрої.

За структурою і організацією доступу до файлів розрізняють **ієрархічні файлові системи** (файли і підкаталоги довільної вкладеності) і **плоскі файлові системи** (набір файлів без підкаталогів). Ієрархічні файлові системи зараз є найбільш поширеними. Існують спеціальні види файлових систем:

– **кластерні файлові системи** дозволяють розподіляти файли між декількома однотипними фізичними пристроями одного комп'ютера;

– **мережеві файлові системи** забезпечують механізми доступу до файлів одного комп'ютера з інших комп'ютерів мережі;

– **розподілені файлові системи** забезпечують зберігання файлів шляхом їх розподілу між декількома комп'ютерами мережі.

Файлову систему зазвичай розглядають як частину операційної системи. Кожна операційна система надає свій набір файлових систем. Наприклад, файлові системи FAT 16, FAT 32, NTFS пов'язані з Windows, Ext2, Ext3, Ext4 використовують у Linux.

1.5. Текстові та бінарні файли

Незалежно від файлової системи, всі файли можна поділити на текстові та бінарні (двійкові).

Текстовий файл (text file) – це комп'ютерний файл, в якому вся інформація подана у вигляді символів визначеної кодової таблиці. Послідовність символів розділена на рядки. Для відокремлення рядків один від одного використовуються роздільники – один або більше спеціальних керуючих символів. Приклади текстових файлів – прості документи, створені за допомогою блокнота (*.txt), початкові коди (сирцеві тексти) програм мовами високого рівня (*.pas, *.c, *.cpp, *.cs, *.java тощо), файли розмітки гіпертексту (*.htm, *.html), форматовані документи (*.rtf) тощо. Підготовка та редагування текстових файлів, незалежно від їх спеціального формату і призначення, може бути здійснена універсальними текстовими редакторами, наприклад, блокнотом.

Двійковий файл (бінарний файл, binary file) – це комп'ютерний файл, в якому числова інформація подана двійковими числами, відповідно до внутрішнього подання у пам'яті комп'ютера (а не символами, як в текстових файлах). Кожен окремий формат двійкового файлу вимагає спеціального програмного забезпечення. Приклади двійкових файлів – виконувані файли (програми) під усіма операційними системами, растрові зображення (*.tif, *.jpeg, *.png, *.gif тощо), архіви (*.zip, *.rar тощо), аудіофайли, відеофайли всіх форматів, файли з двійковим кодом (*.obj, *.class тощо), а також численні спеціальні формати пакетів прикладних програм.

2. Основні поняття програмування

2.1. Мови програмування

Мова програмування – це спеціальна нотація, за допомогою якої можуть бути записані інструкції, що забезпечують керування роботою комп'ютера. Мови програмування поділяють на низькорівневі та високорівневі.

– **низькорівневі мови** призначені для конкретного комп'ютера і віддзеркалюють його машинні коди; крім мови машинних команд, до низькорівневих мов також належить мова Асемблера;

– **високорівневі мови** не залежать від машинного коду конкретного комп'ютера і дозволяють працювати з абстрактними даними.

Мови високого рівня поділяють на основні групи:

– процедурні (FORTRAN, ALGOL-60, BASIC, ALGOL-68, PASCAL, C, Modula-2 тощо);

– об'єктно-орієнтовані (Simula-67, Smalltalk-80, C++, ADA, Object Pascal, OBERON, Java, C# тощо).

Існують також мови функціонального (логічного), декларативного та інших форм програмування.

2.2. Інтерпретатори і компілятори. Етапи розробки програми

Інструкції високорівневої мови програмування повинні бути переведені (трансльовані) у машинний код за допомогою спеціальної програми, яка має назву **транслятора** (translator). Транслятори бувають двох типів – інтерпретатори і компілятори:

– **інтерпретатор (interpreter)** транслює програму рядок за рядком і одразу ж виконує інструкції, зазначені у цих рядках (наприклад, інтерпретатори BASIC, JavaScript); інтерпретатори забезпечують гнучкість і створюють ефект миттєвого виконання, але необхідність багаторазової трансляції раніше інтерпретованих рядків під час виконання істотно знижує ефективність програми;

– **компілятор** (compiler) транслює весь код у набір команд, які може виконати процесор (віртуальна машина). Програми, написані мовами Pascal, C++, C# та багатьма іншими завжди оброблюються компіляторами.

Типовими етапами розробки програми є такі:

– **початковий програмний код** (сирцевий текст, source code) програми готується за допомогою текстового редактора;

– початковий код перетворюється у **набір двійкових інструкцій** (binary code); це може бути машинний код, як у C++, або проміжний двійковий код, як у Java;

– **скомпільований код** збирається з окремих частин (компонування, linking) і виконується.

У випадку виникнення помилок на різних етапах розробки, послідовність кроків повторюється. Виконання програми з метою виявлення помилок і перевірки відповідності алгоритму має назву **зневадження** (debug, отладка).

2.3. Типи застосунків

Прикладні програми (застосунки, applications, приложения), які створюють у сучасних середовищах програмування, з точки зору взаємодії з користувачем можна розділити на дві основні групи:

– консольні застосунки (console applications);

– застосунки графічного інтерфейсу користувача (graphical user interface applications, GUI applications).

Консольні застосунки виконуються у спеціальному консольному вікні, або у повноекранному режимі. Для введення даних використовують клавіатуру (стандартний пристрій введення). Дані вводяться або як аргументи командного рядку (після імені програми), або під час виконання програми.

Для створення консольних застосунків традиційно застосовують концепцію (парадигму) **програмування, керованого даними**. Загальна структура програми включає введення вихідних даних, обчислення та виведення результатів. Під час обчислення, залежно від вихідних даних та проміжних результатів, може здійснюватися розгалуження, виконання циклічних операцій, виклик підпрограм тощо.

Програми графічного інтерфейсу користувача передбачають взаємодію користувача із застосунком через графічні елементи управління (кнопки, позиції меню, елементи списків, лінійки прокрутки тощо) та різноманітні технічні пристрої введення і позиціонування (клавіатура, маніпулятор «миша», джойстик та ін.). Виведення результатів також може здійснюватись у діалогові вікна, у вигляді тексту, графіків тощо.

Складна логіка взаємодії користувача з програмою унеможливлює застосування методів програмування, керованого даними. Сьогодні для програмування застосунків графічного інтерфейсу користувача використовують парадигму **програмування, керованого подіями**. Уся програма складається з ініціалізації (реєстрації візуальних елементів управління) та основного циклу отримання та обробки подій. Події – це

переміщення або натискання кнопок миші, клавіатурне введення тощо. Кожен зареєстрований візуальний елемент управління може отримувати події, які стосуються його роботи, та виконувати функції обробки цих подій. У разі необхідності він може надсилати події іншим зареєстрованим компонентам.

У типовому випадку здійснюється створення одного або декількох вікон, після чого до них додаються візуальні елементи управління. Для цих елементів створюються та реєструються функції обробки певних подій. Основний цикл отримання та обробки подій здійснюється стандартними засобами – за допомогою каркасу застосунку (framework). Розробникові конкретної програми тільки треба додати необхідні елементи та написати функції обробки подій.

3. Алгоритми

Алгоритм (algorithm) – це детальний опис послідовності дій, спрямованих на розв'язання визначеної задачі. Мета алгоритму повинна бути досягнута за скінченну кількість кроків. Алгоритми зазвичай поділяють на лінійні, алгоритми з розгалуженням і циклічні.

Є декілька способів опису алгоритмів, наприклад вербальний спосіб або псевдокод. Найбільш наочним є графічний опис. Існує дві загальноприйняті форми графічного зображення алгоритмів – традиційна блок-схема та діаграма діяльності. Остання форма є частиною так званої Уніфікованої мови моделювання.

Уніфікована мова моделювання (Unified Modeling Language, UML) – це графічна нотація для визначення, опису, проектування та документування програмних систем, бізнес-систем та інших систем різної природи, у першу чергу пов'язаних з програмним забезпеченням. UML включає низку діаграм для моделювання і проектування складних систем.

Діаграма діяльності (Activity diagram) – одна зі стандартних діаграм UML. Цей вид діаграм може бути використаний для зображення алгоритмів. У цьому випадку використовуються такі елементи діаграми:

- початковий стан (initial state);
- діяльність (activity);
- перехід (transition);
- символ перевірки умови (decision);
- кінцевий стан (end state).

На кожній діаграмі діяльності може бути присутнім тільки один початковий стан (рис. 1).

Рисунок 1 – Початковий стан

Будь-яка діяльність (введення, виведення, обчислення та ін.) на діаграмі подається овалом (рис. 2).

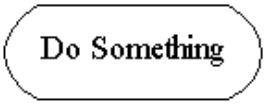

Рисунок 2 – Діяльність

Діяльності зв’язуються одна з одною переходами у вигляді стрілок (рис. 3).

Рисунок 3 – Перехід

Символ перевірки умови (decision) зображують у вигляді ромба. Умови переходу записують у квадратних дужках (рис. 4).

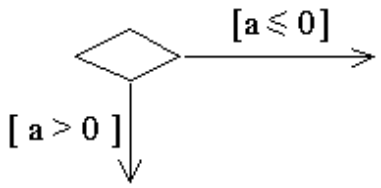

Рисунок 4 – Перевірка умови

Кінцевих станів може бути декілька. Вони зображуються у вигляді чорного кола з обрамленням (рис. 5).

Рисунок 5– Кінцевий стан

З початкового стану виходить одна і тільки одна стрілка. Елемент «діяльність» вимагає, щоб у нього входила одна стрілка, а одна виходила. Не повинно бути діяльностей без вхідної або без вихідної стрілки. Декілька стрілок може виходити тільки із символу перевірки умови.

На рис. 6 наводиться схема алгоритму обчислення середнього значення. Це типовий лінійний алгоритм без розгалуження і циклів, оскільки всі дії виконуються один раз, і за будь-яких умов.

Реалізація алгоритму обчислення зворотної величини вимагає перевірки можливості ділення (аргумент не може приймати значення 0) (рис. 7).

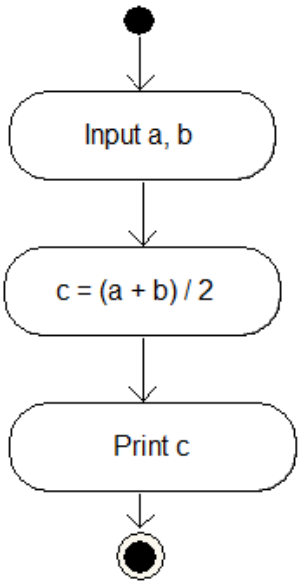

Рисунок 6 – Алгоритм обчислення середнього значення

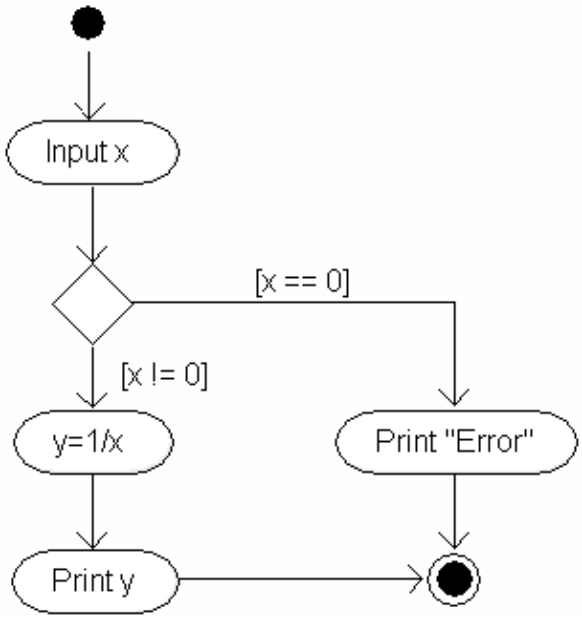

Рисунок 7 – Алгоритм обчислення зворотної величини

Циклічний алгоритм передбачає виконання певних обчислень декілька разів. Сукупність таких дій, що циклічно повторюються, має назву **тіла циклу**. До початку виконання тіла циклу або після завершення дій у тілі циклу слід здійснювати перевірку необхідності повторного входу в цикл.

Алгоритм на рис. 8 описує дії, пов’язані з уведенням цілого *n* та знаходженням степенів числа 2 – від першого до *n*-го.

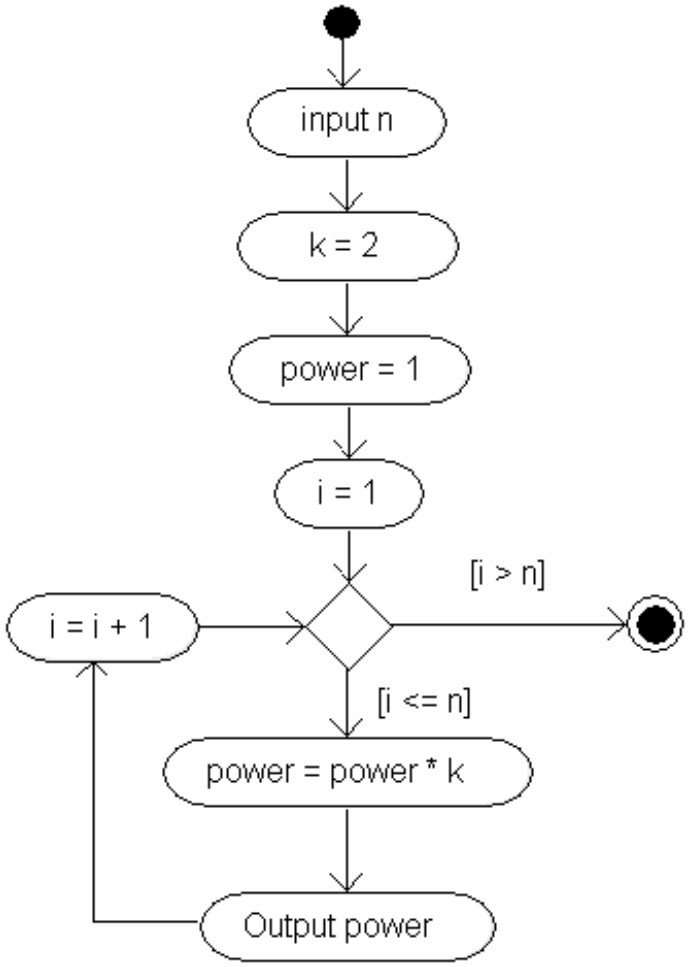

Рисунок 8 – Алгоритм обчислення степенів числа 2

Під час створення алгоритмів за допомогою діаграм діяльності окремі повідомлення можна виводити будь-якими мовами (наприклад, «Print *x*», «Друкувати *x*» тощо), оскільки діаграма не пов’язана з певною мовою програмування та її програмна реалізація виконується вручну. Головне – діаграма повинна бути зрозумілою виконавцеві.

4. Середовище програмування Microsoft Visual Studio .NET

4.1. Створення нового проекту

Microsoft Visual Studio .NET – це комплект (suite) засобів розробки, що включає Visual Basic .NET, Visual C++ .NET, Visual F++ .NET та Visual C# .NET, а також підтримку Web-програмування. Мови розробки використовують загальне **інтегроване середовище розробки** (Integrated Development Environment, IDE). Інтегроване середовище розробки дозволяє програмісту створювати проекти, здійснювати доступ до файлів допомоги, редагувати й компілювати код, виправляти помилки компіляції та компонування.

Після першого завантаження IDE з'являється стартова сторінка (`Start Page`), на якій, зокрема, можна знайти кнопки «Відкрити проект» (`Open Project`) і «Новий проект» (`New Project`).

Ключовим поняттям розробки програми у Visual C++ є **проект** (project). Проект – це набір взаємозалежних вихідних файлів, компіляція і компонування яких дозволяє створити програму або DLL (Dynamic link library). Початкові файли проекту зазвичай зберігаються в окремій теці. Всередині проекту можна створювати декілька **конфігурацій** – узгодженого набору налаштувань, зв'язаних з певною метою діяльності. Кожен проект містить щонайменше дві конфігурації – **конфігурацію налагодження** (debug configuration) і **конфігурацію релізу** (release configuration).

Новий проект може бути створений декількома способами:

– з використанням сторінки `Start Page`;

– з використанням головного меню `FILE | New | Project...`;

– з використанням кнопки `New Project`;

– клавішною комбінацією `Ctrl+Shif+N`.

З'являється вікно майстра нового проекту (`New Project`), в якому слід виконати такі дії:

– серед встановлених типів проектів (Installed) обрати тип проекту Visual C++;

– у середній частині вікна обрати Empty Project;

– ввести ім’я проекту в полі Name; усталене місце розташування для нового проекту визначається автоматично, але у більшості випадків слід змінити диск і теку;

– натиснути OK; порожній проект створений.

Якщо замість Empty Project було обрано шаблон Win32 Project, слід здійснити деякі налаштування у майстрі нового проекту (Project Wizard):

– перейти на закладку Application Settings;

– обрати опції Console Application і Empty project, після чого натиснути Finish.

Примітка: не слід вживати шаблон Win32 Console Application без додаткових налаштувань, оскільки згенерований код міститиме елементи, специфічні для Microsoft Visual Studio, і не буде відповідати стандарту C++.

Щоб створити новий початковий файл, слід виконати такі дії:

– у меню PROJECT обрати функцію Add New Item... (або Add | New Item... у контекстному меню пункту провідника рішень проекту);

– обрати тип файлу: C++ File. Рекомендується ввести ім’я файлу в полі Name; в іншому випадку ім’я буде встановлено у Source.cpp;

– натиснути OK, після чого в панелі редактора відкривається порожній файл.

Тепер можна ввести та зберегти текст програми. Для виконання програми можна використовувати клавішну комбінацію Ctrl-F5 (Start Without Debugging).

4.2 Налагодження програми

Засоби налагодження програми використовуються для пошуку і виправлення логічних помилок.

Крім автоматичного виконання програми, з метою налагодження передбачений покроковий режим. До тексту програми необхідно додати принаймні одну точку переривання (breakpoint). Це можна зробити за допомогою клавіші **F9**, коли курсор знаходиться у необхідному рядку. Коли програма виконується у режимі зневадження, точка переривання обумовлює тимчасову зупинку програми. Далі можна зняти програму з використання, або здійснювати звичайне чи покрокове виконання.

Після встановлення точок переривання можна завантажити програму в режимі налаштування (`DEBUG | Start Debugging` або **F5**). Після зупинки у точці переривання можна здійснювати покрокове виконання або продовжити виконання далі (до кінця або до наступної точки переривання). Для продовження у звичайному (не покроковому) режимі використовують **F5**. Покрокове виконання програми може відбуватися із заходженням у підпрограми (`DEBUG | Step Into`, функціональна клавіша **F11**) і з пропуском (непокроковим виконанням) підпрограм (`DEBUG | Step Over`, функціональна клавіша **F10**).

Якщо під час зневадження помістити курсор миші на змінну, її значення можна подивитись у маленькому віконці. Можна також скористатися засобами відображення та редагування в спеціальних підвікнах.

Підвікно `Find Symbol Results` містить такі закладки:

– закладка `Autos` відображає інформацію про змінні, які вживані в поточній та попередній інструкціях;

– закладка `Locals` відображає інформацію про змінні, які є локальними всередині поточної функції;

– закладка `Watch 1` дозволяє переглядати і редагувати значення, які зберігаються в змінних.

Якщо у вікні `Locals` відображається змінна або структура, поруч з її ім'ям з'являється кнопка. Натиснувши на кнопку, можна розширити або звузити перегляд змінної. Кнопка має вигляд знаку плюс (+), якщо змінна відображається в скороченому вигляді, і мінус (-), коли вона відображається в розгорнутому вигляді.

Достроково припинити виконання можна за допомогою функції `DEBUG | Stop Debugging` (Shift-F5). Можна також зупинити програму з одночасним завантаженням і виконанням з початку (`DEBUG | Restart` або Ctrl-Shift-F5).

ПРИКЛАДИ ПРОГРАМ

Номери рядків у наступних прикладах програм не є частиною коду. Ці числа можуть бути видалені разом з коментарями, в яких вони розташовані. Синтаксичні конструкції, вживані у лекціях, будуть детально роз'яснені у подальших темах.

1. Середня швидкість. Нехай необхідно написати програму, яка зчитує значення відстані між двома містами і тривалість поїздки та обчислює середню швидкість. Програма реалізовуватиме лінійний алгоритм, аналогічний наведеному на рис. 1.

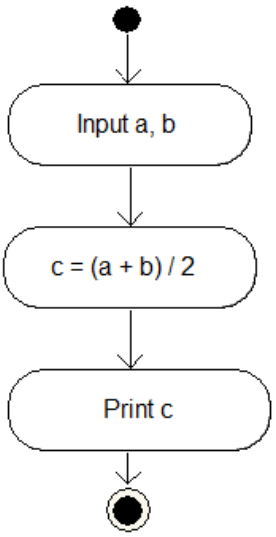

Рисунок 1 – Алгоритм обчислення середнього значення

Можна створити таку програму:

```
/* 01 */  // Середня швидкість
/* 02 */  #include <iostream>
/* 03 */  using namespace std;
/* 04 */  int main(int argc, char* argv[])
/* 05 */  {
/* 06 */      setlocale(LC_ALL,"UKRAINIAN");
/* 07 */      float s, t;
/* 08 */      cout << "Уведіть відстань і час:" << endl;
/* 09 */      cin >> s >> t;
/* 10 */      float v = s / t;
/* 11 */      cout << "Швидкість: " << v << endl;
/* 12 */      return 0;
/* 13 */ }
```

Рядок 01 містить **коментар** – деякий пояснювальний текст усередині тексту програми; коментар призначений для того, щоб допомогти людині зрозуміти програмний код. Коментарі – це просто текст, який ігнорується компілятором.

Рядок 02 – це так звана **директива препроцесору**. Препроцесор – це підпрограма, обробляє початковий текст відповідно до її директив і створює **одиницю трансляції** (translation unit). Виконуючи директиву `include`, препроцесор вставляє текст указаного **заголовного файлу** в одиницю трансляції. Стандартний заголовний файл `iostream` містить, зокрема, оголошення стандартного потоку введення `cin` (клавіатура), стандартного потоку виведення `cout` (консольне вікно), і так званого маніпулятора `endl` (кінець рядка).

У рядку 03 здійснюється підключення так званого простору імен. У цьому випадку підключається простір імен `std`. Таке підключення забезпечує можливість використовувати у тексті імена `cin`, `cout` і `endl` безпосередньо, а не `std::cin`, `std::cout` і `std::endl`.

Примітка: підключення всіх імен певного простору може призвести до конфліктів імен і тому є прийнятним тільки у невеличких навчальних програмах.

Рядок 04 містить **заголовок функції main()**. Виконання будь-якої програми починається з першої інструкції функції з ім'ям `main()`. Дужки охоплюють список аргументів. Цілий аргумент `argc` показує кількість параметрів командного рядка. Аргумент `argc` – це вказівник на масив параметрів командного рядка. Якщо непотрібно обробляти аргументи командного рядка, функцію `main()` можна визначити без аргументів:

```
int main()
{

}
```

Фігурні дужки у рядках 05 і 13 охоплюють програмний блок функції `main()`. Програмний блок містить інструкції, які будуть виконані.

У рядку 06 викликається стандартна функція `setlocale(LC_ALL, «UKRAINIAN»)`, яка забезпечує використання відповідної кодової таблиці під час консольного виведення.

Примітка*:* навіть якщо правильно вказати локалізацію, можуть виникнути проблеми з відображенням української літери і. В усіх прикладах замість української літери доводиться вживати відповідну літеру латинської абетки.

Твердження в рядку 07 – це визначення двох змінних типу `float`. Цей тип використовують для подання дійсних даних. Змінна s зберігатиме відстань, а t – час.

У рядку 08 послідовність символів у подвійних лапках `"Уведіть відстань і час:"` записується у стандартний вихідний потік виведення `cout`. Під час виконання оператора на екран виводиться текст рядка (без

лапок). Занесення у потік виведення маніпулятора `endl` забезпечує перехід курсора на новий рядок.

У рядку 09 подана операція читання значень *s* і *t* зі стандартного потоку введення `cin` (у цьому випадку – з клавіатури). У рядку 10 визначення змінної *v* об'єднане з обчисленням виразу *s / t*. У рядку 11 на екран виводиться константа "Швидкість: " та значення змінної *v*.

Значення, яке повертає функція `main()` у рядку 12, може бути використане операційною системою. Значення 0 свідчить про нормальне завершення.

Примітка. Якщо завантажити програму на виконання без зневадження (`Ctrl-F5`), після її завершення у консольному вікні з'явиться додатковий рядок «`Press any key to continue . . .`». Закінчити роботу можна, натиснувши будь-яку клавішу. Якщо рядок не з'являється, наприклад, через запуск зі зневадженням (**F5**), до програми можна додати виклик функції, яка очікує на натиснення будь-якої клавіші перед виходом з функції:

```
system("pause"); // З'явиться "Press any key to continue..."
return 0;
```

2. Цілий степінь. Наступна програма передбачає введення основи степеня і цілого показника степеня, обчислення і виведення степеня (циклічний алгоритм, аналогічний наведеному на рис. 2):

```
/* 01 */  // Цілий степінь
/* 02 */  #include <iostream>
/* 03 */  using namespace std;
/* 04 */  int main()
/* 05 */  {
/* 06 */      setlocale(LC_ALL,"UKRAINIAN");
/* 07 */      float x;
/* 08 */      int n;
/* 09 */        cout << "Уведіть основу степеня і показник
степеня:" << endl;
/* 10 */      cin >> x >> n;
/* 11 */      float power = 1;
/* 12 */      for (int i = 1; i <= n; i++)
```

```
/* 13 */          power *= x;
/* 14 */      cout << "Степінь: " << power << endl;
/* 15 */      return 0;
/* 16 */  }
```

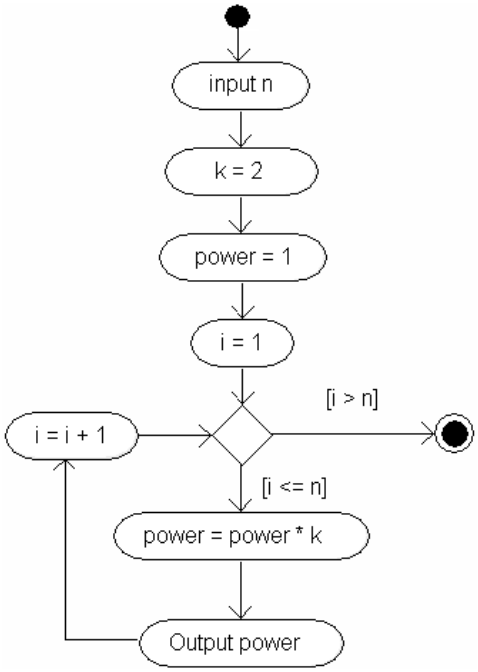

Рисунок 2 – Алгоритм обчислення степенів числа 2

На відміну від основи степеня (`x`), показник степеня *n* визначається в рядку 08 як ціла змінна.

У рядку 11 визначається змінна `power` типу `float` і їй присвоюється значення 1. Це значення використовується як перше наближення результату.

Рядок 12 містить оператор циклу `for` – циклу з параметром. Параметр `i` визначається всередині циклу і змінюється від 1 (`int i = 1`) з кроком 1 (`i++`) поки виконується умова `i <= n` (менше або дорівнює). На кожному кроці циклу виконується інструкція, що міститься безпосередньо після інструкції `for (...)`.

Рядок 13 є «тілом циклу». У ньому *n* разів здійснюється домноження значення змінної `power` на значення `x` (у комірку, що містить спочатку 1, `n` разів записується значення, рівне попередньому, помноженому на `x`).

ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ

1. Реалізація алгоритму з розгалуженням

Реалізувати алгоритм розв'язання квадратного рівняння. Алгоритм повинен включати всі можливі варіанти вихідних даних.

2. Реалізація циклічного алгоритму

Реалізувати алгоритм обчислення виразу:

$$y = \frac{1}{x+2} + \frac{2}{x+4} + \ldots + \frac{k-1}{x+2(k-1)} + \frac{k+1}{x+2(k+1)} + \frac{n}{x+2n}.$$

Забезпечити перевірку можливих помилок.

3. Індивідуальне завдання

Розробити алгоритм програми, яка обчислює значення функції у заданому діапазоні. Програма повинна прочитати значення початку і кінця інтервалу, крок збільшення аргументу і значення *n*. Алгоритм повинен бути поданий з використанням UML-діаграми діяльності. Алгоритм повинен містити такі частини:

– зчитування даних;

– основний цикл, в якому встановлюється нове значення аргументу, розраховується значення функції, виводяться на екран значення аргументу і функції та здійснюється збільшення значення аргументу на величину кроку.

Конкретна функція визначається відповідно до номера у списку студентів у групах (номер варіанта) (табл. 1).

Таблиця 1 – Варіанти для виконання індивідуального завдання

№	Завдання	№	Завдання
1	$y = \begin{cases} \sum_{i=1}^{n} (i+x)^2, x < 0 \\ \sum_{i=0}^{n-1} \prod_{j=1}^{n} \frac{x+i}{i+j}, x \geq 0 \end{cases}$	9	$y = \begin{cases} \sum_{i=2}^{n-1} \frac{x}{i}, x \leq 0 \\ \sum_{i=0}^{n-1} \sum_{j=0}^{i} \frac{i}{j+x}, x > 0 \end{cases}$

Продовження таблиці 1

№	Завдання	№	Завдання
2	$y=\begin{cases}\sum\limits_{i=1}^{n-1}\sum\limits_{j=1}^{n}(x-i+j), x<0\\ \sum\limits_{i=0}^{n-1}\frac{x}{i}, x\ge 0\end{cases}$	10	$y=\begin{cases}\sum\limits_{i=1}^{n}\sum\limits_{j=1}^{n}\frac{1}{x-i-j}, x<0\\ \prod\limits_{i=0}^{n-3}(x-i), x\ge 0\end{cases}$
3	$y=\begin{cases}\prod\limits_{j=2}^{n-2}(j+1), x<0\\ \sum\limits_{i=0}^{n-1}\prod\limits_{j=0}^{n-1}\left(x+i+j^2\right), x\ge 0\end{cases}$	11	$y=\begin{cases}\sum\limits_{i=0}^{n-1}\sum\limits_{j=0}^{n-1}\frac{1}{x-i-j}, x\le 0\\ \prod\limits_{i=1}^{n}\left(\frac{1}{x}-\frac{1}{i}\right), x>0\end{cases}$
4	$y=\begin{cases}\prod\limits_{j=0}^{n-1}(i^2+i), x\le 0\\ \sum\limits_{i=1}^{n-1}\prod\limits_{j=0}^{n-1}\frac{x}{i+j}, x>0\end{cases}$	12	$y=\begin{cases}\sum\limits_{i=0}^{n}\prod\limits_{j=1}^{n-1}\left(i^2+j\right), x<0\\ \sum\limits_{i=1}^{n-2}(i-x), x\ge 0\end{cases}$
6	$y=\begin{cases}\sum\limits_{i=0}^{n}(x-i)^2, x\le 0\\ \prod\limits_{i=1}^{n}\prod\limits_{j=0}^{n-1}(x-i-j), x>0\end{cases}$	14	$y=\begin{cases}\prod\limits_{j=2}^{n-2}j^2, x<0\\ \sum\limits_{i=0}^{n-1}\prod\limits_{j=0}^{n-1}\left(x+i^2+j\right), x\ge 0\end{cases}$
7	$y=\begin{cases}\sum\limits_{i=0}^{n}\prod\limits_{j=1}^{n-1}(i-j), x<0\\ \sum\limits_{i=1}^{n-2}(i-x)^2, x\ge 0\end{cases}$	15	$y=\begin{cases}\sum\limits_{i=2}^{n-1}\frac{x-1}{i}, x\le 0\\ \sum\limits_{i=0}^{n-1}\sum\limits_{j=0}^{i}\frac{i+1}{j+x}, x>0\end{cases}$
8	$y=\begin{cases}\sum\limits_{i=1}^{n-1}\sum\limits_{j=1}^{n}\left(x-i^2+j\right), x<0\\ \sum\limits_{i=0}^{n-1}\frac{x-1}{i+1}, x\ge 0\end{cases}$	16	$y=\begin{cases}\prod\limits_{i=0}^{n-1}(i+i), x\le 0\\ \sum\limits_{i=0}^{n-1}\sum\limits_{j=1}^{n-1}\frac{x+j}{i+j+1}, x>0\end{cases}$

ВПРАВИ ДЛЯ КОНТРОЛЮ

Завдання 1. Розробити алгоритм та написати програму, в якій здійснюється читання значення певної довжини у дюймах і обчислюється й виводиться значення цієї довжини у міліметрах (1 дюйм = 25,4 мм).

Завдання 2. Розробити алгоритм та написати програму, яка зчитує вісім значень і повертає середнє арифметичне.

Завдання 3. Розробити алгоритм та написати програму, яка зчитує значення змінної *n* цілого типу й обчислює *n!*.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Що таке позиційна система числення?
2. Що таке основа системи числення?
3. Які є недоліки й переваги двійкової системи числення?
4. Для чого використовують шістнадцяткову систему числення?
5. Визначте поняття програмного забезпечення.
6. Як можна класифікувати програмне забезпечення?
7. Що таке інструментальні засоби?
8. Що таке застосунок (додаток)?
9. Які функції виконує операційна система?
10. Наведіть приклади операційних систем.
11. Що таке файлова система?
12. Чим текстові файли відрізняються від бінарних?
13. Наведіть приклади текстових і бінарних файлів.
14. Чим визначається «рівень» мови програмування?
15. Як визначити поняття «комп’ютерна програма»?
16. Які існують етапи розробки програми?
17. Що таке зневадження?
18. Що таке консольний застосунок і чим він відрізняється від інших видів застосунків?

19. У чому переваги і недоліки інтерпретаторів і компіляторів?

20. Що таке алгоритм?

21. Які є способи подання алгоритму?

22. Які є різні типи алгоритмів?

23. Що таке UML?

24. Що таке діаграма UML?

25. У чому різниця між блок-схемою і діаграмою діяльності?

26. У чому різниця між поданням умовних елементів у блок-схемі та діаграмі діяльності?

27. У чому різниця між поданням виведення та розрахунків у нотації діаграм діяльності?

28. Що таке інтегроване середовище розробки?

29. Які мови програмування підтримує Visual Studio?

30. Що таке зневадження?

31. Як створити новий проект у Visual Studio?

32. Як зневаджувати програми у Visual Studio?

Рекомендована література

1. Bjarne Stroustrup. The C++ Programming Language. Third Edition / Bjarne Stroustrup. – Addison-Wesley, 1997.

2. Stanley B. Lippman C++ Primer. Third Edition / Stanley B. Lippman, Josee Lajoie. – Addison-Wesley, 1988.

3. Deitel H. M. C++. How to Program. Third Edition / H. M. Deitel, P. J. Deitel. – Prentice Hall, 2001.

4. Голуб Б. М. C#. Концепція та синтаксис / Б. М. Голуб. – Львів: Видавничий центр ЛНУ ім. Івана Франка, 2006. – 136 с.

5. Грицюк Ю. І. Програмування мовою С++: навч. посібник / Ю. І. Грицюк, Т. Є. Рак. – Львів : Вид-во Львівського ДУ БЖД, 2011. – 292 с.

Internet-джерела

1. The C++ Programming Language (Bjarne Stroustrup's homepage) // http://www2.research.att.com/~bs/C++.html

2. ISO/IEC 14882:2003 Programming languages - C++ (International Stamdard) // http://cs.nyu.edu/courses/summer12/CSCI-GA.2110-001/downloads/C++%20Standard%202003.pdf

3. The C++ Resources Network // http://www.cplusplus.com/

4. The C++ Tutorial // http://www.learncpp.com/

5. C++ – Вікіпідручник // http://uk.wikibooks.org/wiki/C++

6. C++ – Вікіпіедія // http://uk.wikipedia.org/wiki/C++

7. Корх О. Основи мови програмування С++ // http://korkholeh.googlepages.com/cppfund.pdf

Навчальне видання

Методичні вказівки

до виконання лабораторної роботи 1

за темою «Основні концепції програмування. Розробка та графічне подання алгоритмів»

з курсу «Алгоритмізація та програмування. Частина 1»

для студентів спеціальності

122 «Комп'ютерні науки»

Укладачі:

ІВАНОВ Лев Вадимович

БІЛОВА Марія Олексіївна

Відповідальний за випуск М. Д. Годлевський

Роботу до видання рекомендував О. В.Горілий

План 2018 р., поз. 310

Підписано до друку 28.05.2019. Гарнітура Times New Roman.

Ум. друк, арк. 1,7.

Видавничий центр НТУ «ХПІ»,

вул. Кирпичова, 2, м.Харків-2, 61002

Свідоцтво про державну реєстрацію ДК № 3478 від 21.08.2017 р.

Самостійне електронне видання

30